The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters*

Brett Bode, David M. Halstead, Ricky Kendall, and Zhou Lei
Scalable Computing Laboratory, Ames Laboratory, DOE
Wilhelm Hall, Ames, IA 50011, USA, help@scl.ameslab.gov
David Jackson, Maui High Performance Computing Center

Abstract

The motivation for a stable, efficient, backfill scheduler that runs in a consistent manner on multiple hardware platforms and operating systems is outlined and justified in this work. The combination of the Maui Scheduler and the Portable Batch System (PBS), are evaluated on several cluster solutions of various size, performance and communications profiles. The total job throughput is simulated in this work, with particular attention given to maximizing resource utilization and to the execution of large parallel jobs.

1 Introduction

With the ever increasing size of cluster computers, and the associated demand for a production quality shared resource management system, the need for a policy based, parallel aware, batch scheduler is beyond dispute. To this end the combination of a stable, portable resource management system, coupled to a flexible, extensible, scheduling policy engine will be presented and evaluated in this work. Features, such as extensive advanced reservation, dynamic prioritization, aggressive backfill, consumable resource tracking and multiple fairness policies, will be defined and illustrated on commodity component cluster systems. The increase in machine utilization and operational flexibility will be demonstrated for a non-trivial set of resource requests over a range of duration, and processor count tasks.

We will use the term *large* to describe jobs that require a substantial portion (>50%) of the available CPU resources of a parallel machine. The duration of a job, is considered to be independent of its resource request, and for the purposes of this paper the term *long* will be used to identify jobs with an extended runtime of multiple hours. The opposite terms of small and short will be used for the converse categories of jobs respectively.

2. Scheduling Terminology

To clarify the nomenclature used in the descriptive sections of this work we will now include a glossary of terms, together with a brief explanation of their meaning.

Utilization and turnaround

The ultimate aim of any dynamic resource administration hierarchy is to maximize utilization and job throughput and minimize turnaround time. The aim of improving utilization can be achieved by allocating tasks to idle processors, but the task of maximizing throughput is much more nefarious, involving complex fair access decisions based on machine stakeholder rights.

Prioritization and fairness

It is the goal of a schedule administrator to balance resource consumption amongst competing parties and implement policies that address a large number of political concerns. One method of ensuring appropriate machine allocation is with system partitioning. This approach, however, leads to fragmentation of the system, and a concomitant fall in utilization efficiency. To be preferred is a method by which the true bureaucratic availability requirements can be met, without negatively impacting the utilization efficiency of the resource.

Fair share

The tracking of historical resource utilization for each user results in the ability to modify job priority, ensuring a balance between appropriate access, and maximizing machine utilization. Users can be given usage targets, floors and

ceilings which can be configured to reflect the budgeted allocation request.

Reservation

The concept of resource reservation is essential in constructing a flexible, policy based batch scheduling environment. This is usually a data structure that defines not only the computational node count, but also the execution timeframe and the associated resources required by the job at the time of its execution.

Resource manager

The resource manager coordinates the actions of all other components in the batch system by maintaining a database of all resources, submitted requests and running jobs. It is essential that the user is able to request an appropriate computational node for a given task. Conversely, in a heterogeneous environment, it is important that the resource manager conserve its most powerful resources until last, unless they are specifically requested. It also needs to provide some level of redundancy to deal with computational node failure and must scale to hundreds of jobs on thousands of nodes, and should support hooks for the aggregation of multiple machines at different sites.

Job scheduler/ Policy manager

The job scheduler takes the node and job information from the resource manager and produces a list sorted by the job priority telling the resource manager when and where to run each job. It is the task of the policy manager to have the flexibility to arbitrate a potentially complex set of parameters required to define a fare share environment, yet retain the simplicity of expression that will allow the system administrators to implement politically driven resource allocations.

Job execution daemon

Present on each node, the execution daemon is responsible for setting up the node, servicing the initiation request from the resource manager, reporting its progress, and cleaning up after the job termination, either upon completion or when the job is aborted. It is important that this be a lightweight and portable daemon, allowing for rapid access to system and job status information and exhibit a low overhead of task initiation, to facilitate scalable startup on massively parallel systems.

Co-allocation

A requirement for an integrated computational service environment is that mobile resources, such as software licenses and remote data access authorizations, may be reserved and accessed with appropriate privileges at execution time. These arbitrary resources need to be made transparently available to the user, and be managed centrally with an integrated resource request notification system.

Meta-scheduling

A meta-scheduler is a technique of abstraction whereby complex co-allocation requests and advanced reservation capabilities can be defined and queried for availability before the controlling job begins execution. This concept ties in naturally to the requirement for data pre-staging since this can be considered as merely another resource.

Pre-staging

The problems of coordinating remote data pre-staging or access to hierarchical storage can be obviated by an intelligent advance reservation system. This requires integration with the meta-scheduler and co-allocation systems to ensure that the initiation of the setup phase is appropriately timed to synchronize with the requested job execution.

Backfill scheduling

The approach of backfill job allocation is a key component of the Maui scheduler. It allows for the periodic analysis of the running queue and execution of lower priority jobs if it is determined that their running will not delay jobs higher in the queue. This benefits short, small jobs the most, since they are able to pack into reserved, yet idle, nodes that are waiting for all of the requested resources to become available.

Shortpool policy

The shortpool policy is a method for reserving a block of machines for expedited execution and turnaround. This is usually implemented during workday hours and can predictively assign currently busy nodes to the shortpool if their task

will finish within the required window (usually under two hours).

Allocation bank

The concept of an allocation bank is essential in a multi-institution shared resource environment [1]. It allows for a budgeted approach to resource access, based on a centralized user account database. The user account is debited upon the execution of each job, with the individual being unable to run jobs once the account has been exhausted.

Reservation security

An essential area of research centers on authentication and security of remotely shared resources. Issues such as secure interactive access and user authentication have been addressed and resolved to a large extent, but the issue of delayed authentication and inter-site trust are still subjects for research. The important factor of non-repudiation needs to be addressed in order to validate the source of a submission.

Job expansion factor

This is a way of giving small time limit jobs a priority over larger jobs by calculating their priority from the sum of the current queue wait time and the requested wall time relative to the requested wall time.

Job Efficiency

This is the percent of the requested time actually used by the job. For simple schedulers this factor has little effect on the overall scheduling performance. However, for the backfill portion of the Maui Scheduler this factor has a much more significant influence, since low job efficiencies cause inaccurate predictions of holes in the reservation schedule. The best ways to improve this factor are user education and resource monitoring.

Quality of service

The Maui Scheduler allows administrators fine grain control over QOS levels on a per user, group and account basis. Associated with each QOS is a starting priority, a target expansion factor, a billing rate and a list of special features and policy exemptions. This can be used impose

graduated access to a limited resource, while ensuring maximum utilization of idle computational assets.

Downtime scheduling

One important advantage of a time based reservation system is that scheduling down-time for repair or upgrade of running components is easily performed. This is convenient for the current clusters, but will become essential as the size and production nature of parallel clusters continues to increase.

SMP aware queuing

The debate over the utility of multiple processors per computational node continues to rage. It is clear, however, that the incremental cost of additional CPUs in a node is less than a concomitant increase in the total number of nodes. The question is how to exploit the co-location advantage of data between SMP CPUs, and to expose this potential performance enhancement to the user in a consistent manner via the scheduling interface. There are several different approaches available to exploit SMP communications (Pthreads, Shmem etc.), but this topic is beyond the purview of this work.

3. Testbed Hardware

3.1 Linux 64 node

The largest test environment considered in this work is a cluster of 64 Pentium Pro machines with 256 MBytes of RAM, connected by a flat 44 Gbit/sec Fast Ethernet switch. The cluster was constructed in accordance to the Scalable Cluster Model [2] with the file server and external gateway node utilizing a dedicated Gigabit Ethernet connection to the switch for improved performance. The compute nodes are running a patched 2.2.13 Linux kernel with the server nodes providing both Fortran and C compilers with MPI and PVM message passing libraries. Version 2.2p11 of the PBS server runs from the file server node, along with the Maui scheduler version 2.3.2.12.

3.2 Compaq 25 node

The other cluster from which results will be reported consists of 25 Compaq 667 MHz Alpha XP1000 machines, 15 of which have 1024 MB

of RAM and 10 have 640 MB of RAM. One of the 25 nodes has reduced scratch disk space and is thus limited to small jobs. These machines are connected by Fast Ethernet and run the Tru64 Unix operating system. This configuration provides a fairly challenging test for the scheduler since there are three different node resource levels available for users to request. Since we ported the Maui Scheduler PBS plugin to Tru64 Unix several months ago, the cluster has been running in production mode, executing parallel computational chemistry jobs using the GAMESS [3] code. We will be using this cluster to illustrate the pitfalls of real-world environments, and to highlight some of the measures that can be taken to ameliorate certain user shortcomings.

4. Batch scheduler description

4.1 Background

Since clusters and cluster-like systems have been around for several years, there have been multiple queuing systems tried out and several are currently in wide use. Among these are the Distributed Queuing System (DQS), Load Sharing Facility (LSF), IBM's LoadLeveler, and most recently the Portable Batch System (PBS). Each of these systems has strengths and weaknesses. While each of these works adequately on some systems, none of them were designed to run on cluster computers. Currently the system with the best cluster support is PBS. Thus we will consider PBS using its built-in scheduler compared with the addition of the plugin Maui scheduler.

4.2 PBS

Portable Batch System is a POSIX compliant batch software processing system originally developed at NASA's Ames research center for their large SMP parallel computers [4]. It has the advantage of being configurable over a wide range of high power computer architectures, from heterogeneous clusters of loosely coupled workstations, to massively parallel supercomputers. It supports both interactive and batch mode, and has a user friendly graphical user interface.

Recently the focus of development has shifted to clusters and basic parallel support has been added. In addition, the Maui scheduler has been ported to act as a plugin scheduler to the PBS system. This combination is proving successful at scheduling jobs on parallel systems. However, since PBS was not designed for a cluster-like computer, it lacks many important features. For instance, while the resource manager and scheduler are able to reserve multiple processors for a parallel job, the job startup, including the administrative scripts, is performed entirely on one node.

PBS includes several built-in schedulers, each of which can be customized for the local site requirements. The default is the FIFO scheduler that, despite its name, is not strictly a FIFO scheduler. The behavior is to maximize the CPU utilization. That is, it loops through the queued job list and starts any job for which fits in the available resources. However, this effectively prevents large jobs from ever starting since the required resources are unlikely to ever available. To allow large jobs to start, this scheduler implements a "starving jobs" mechanism. This mechanism initiates when a job has been eligible to run (i.e. first in the queue) longer than some predefined time (the default is 24 hours). Once the mechanism kicks in, the scheduler halts starting of new jobs until the "starving" job can be started. It should be noted that the scheduler will not even start jobs on nodes which do not meet the resource requirements for the "starving job".

4.3 Maui

Perhaps the most complete system currently available is the IBM LoadLeveler software developed originally for IBM's SP machines, but now available on several platforms (Linux is not supported). While LoadLeveler provides many useful features, its implementation leaves a lot to be desired. For instance the scheduler was immediately recognized as inadequate, since its poor parallel scheduling resulted in low total machine usage due to many idle nodes waiting for future jobs. To solve this problem the Maui Scheduler [5] was written principally by David Jackson for the Maui High Performance Computer Center. This scheduler has proven to be a dramatic success on the SP platform, so much so that it is now used in place of the default scheduler in LoadLeveler at many SP installations. LoadLeveler is probably the only currently available package, which was designed for a parallel computer from the beginning and thus addresses many of the requirements listed

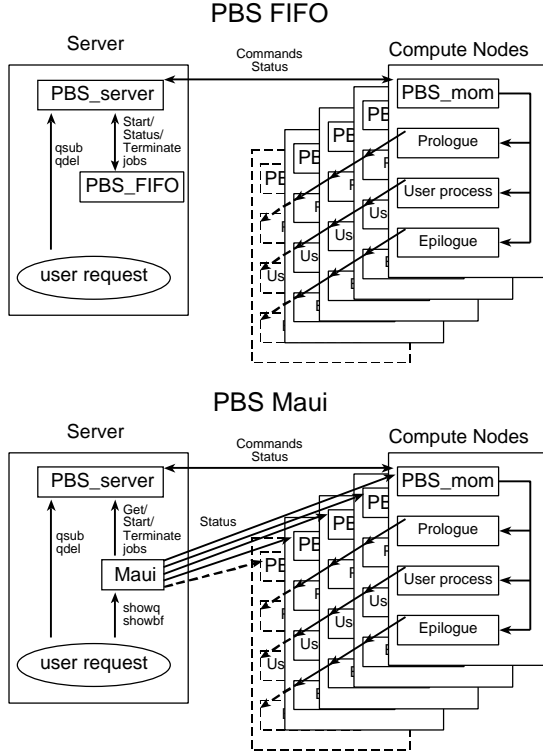above. Figure 1 illustrates the difference between the PBS FIFO and PBS with the Maui scheduler in place.

Figure 1. Schematic of the interaction profile between PBS running the FIFO scheduler and the Maui Scheduler.

The key to the Maui Scheduler is its wall-time based reservation system. This system orders the queued jobs based upon priority (which in turn is derived from several configurable parameters), starts all the high priority jobs that it can, and then makes a reservation in the future for the next high priority job. Once this is done, the backfill mechanism attempts to find lower priority jobs that will fit into time gaps in the reservation system. This gives large jobs a guaranteed start time, while providing a quick turn around for small jobs.

5. Evaluation description

5.1 The Simulated Job Mix

The right mix of jobs for any simulation is nebulous at best. Nothing is better than a real job mix from the user community in question, but that is impossible to reproduce due to user dynamics. User resource requests vary directly with their needs and cycle with external forces such as conference deadlines. To this end we have defined a job mix that fits a rough average of what we have observed on our research clusters and on the MPP systems available at supercomputer centers such as NERSC [6].

The job mix has Large, Medium, Small, Debug, and Failed jobs. Each job has a randomized set of the number of processors (nproc), the time actually spend doing work (work time), the time requested from the resource management system (submit time) and a submission delay time (delay time). Large, Medium, and Small jobs have a work time that is 70% or more of the submit time. Submit time is always greater than or equal to the work time. Large jobs are those that have nproc > 50% of those available. Medium jobs have nproc between 15% and 50% of the available nodes. Small jobs are those with nproc between 30% and 15% of available nodes. Debug jobs have a work time that is greater than 40% of the submit time but use less than 10% of the available processors. Failed jobs are defined by a work time that is less than 20% of the submit time.

Close inspection of these parameters will show that not all jobs generated by a randomized nproc, work time, and submit time, fall into these categories. One further target constraint is that Large jobs are 30% of the total set of jobs, Medium jobs are 40%, Small jobs are 20%, Debug Jobs and Failed jobs are both 5%. Jobs are randomly generated and then classified as Large, Medium, Small, Debug, Failed or "undefined" jobs. Undefined jobs are automatically rejected and others are added only if their addition will not increase their constrained classification above the limits outlined above.

In the 76 jobs of the job mix used in these simulations, 480 random jobs were generated. The resultant mix from this defined job mix algorithm yielded 28.95% Large, 40.79% Medium, 19.74% Small, and 5.26% Debug and Failed jobs. In actual numbers this corresponds to 22 Large, 31 Medium, 15 Small, 4 Debug, and 4 Failed jobs.

The randomized delay time has the effect of jobs being submitted in a random order to the batch system. The first job generated will not necessarily be the first job submitted. All of the

job mix data is available online [7]. The exact same job mix was used with each scheduler setup, PBS/FIFO, PBS/MAUI and PBS/MAUI with backfill turned off.

5.2 Users and the Job Mix.

The defined job mix does not consider user interaction currently. There are no automatic or post submitted jobs that fit any gaps in the system as the scheduler runs jobs, e.g., jobs to fit the backfill mechanism available in some schedulers. Furthermore, it is quite typical for users to simply submit jobs with the maximum allowed time for the queue in question. Our simulation assumes users can predict the resources needed with reasonable accuracy. All jobs are submitted after 180 minutes from the start of the simulation. This does not match the constant influx of jobs on our research cluster or at any supercomputer center.

6. Test results

6.1 Simulation results

Perhaps the most significant result and the simplest are the total run time for each of the scheduler configurations as shown in Table 1.

Table 1.

Scheduler	Total run time (Hours)
PBS FIFO	71.1
Maui Scheduler	66.75
Maui without backfill	66.71
Theoretical Minimum	53.6
Sequential Maximum	90.2

Table 1 includes a Theoretical Minimum time which is simply the total number of node-wall hours divided by 64 (the number of available CPUs). This is clearly not an achievable value since it ignores the packing efficiency of the jobs. Conversely the Sequential Maximum represents the maximum total time the tests would take if they were simply run in a FIFO fashion with no attempt to overlap jobs.

Obviously both schedulers do substantially better than the Sequential Maximum, and the Maui scheduler does substantially better than the PBS FIFO scheduler. It may, however, be surprising that the backfill scheduler is actually slower than Maui without backfill even if only by a small amount. This can be explained by the job efficiencies, which for the test set of jobs had all but 5 jobs with an efficiency of 50% greater and 23 of the 76 jobs with an efficiency greater than 90%. If all jobs had an efficiency of 100% then the backfill algorithm would always be faster. However, since most jobs finish significantly before their scheduled end time it is possible that a backfilled job will keep a reservation from being started early. It is important to note that backfilled jobs will never prevent a job reservation from starting on time, but it might prevent a job reservation from moving forward in time.

Figure 2 illustrate the quite different ways in which the Maui Scheduler and the FIFO scheduler operate. The upper frame shows time, in hours, on the x-axis and job sequence number on the y-axis for the FIFO scheduler. The job sequence number represents the order in which the jobs were submitted to the system. The lower half shows the same information for the Maui Scheduler. Since the Maui Scheduler result without the backfill algorithm was so similar to the regular Maui Scheduler result, it was not plotted separately.

For the FIFO scheduler the job profile shows that initially mainly small jobs were run up until the starving job state kicked in for the for the first large job in the queue. Once in the starving job state FIFO became truly a first in first out scheduler.

The profile for the Maui scheduler is certainly anything but FIFO. It shows a more uniform queue wait time that is driven more by the number of nodes requested than by the initial queue order. This results in the smaller node requests being run along with the larger node requests rather than all at once as with the FIFO scheduler. Because of this the Maui Scheduler is able to maintain higher average node utilization during the first portion of the test run, until it runs out of small node requests to backfill. This effect is illustrated in Figure 2 that shows the node utilization over the test run for all three scheduler tests. Figure 3 shows that Maui is able to maintain a more consistently high node utilization until about halfway through the test when it ran out of small jobs. The FIFO scheduler started out high, but then suffered a large dip as it cleared out the small jobs to let a large job start.
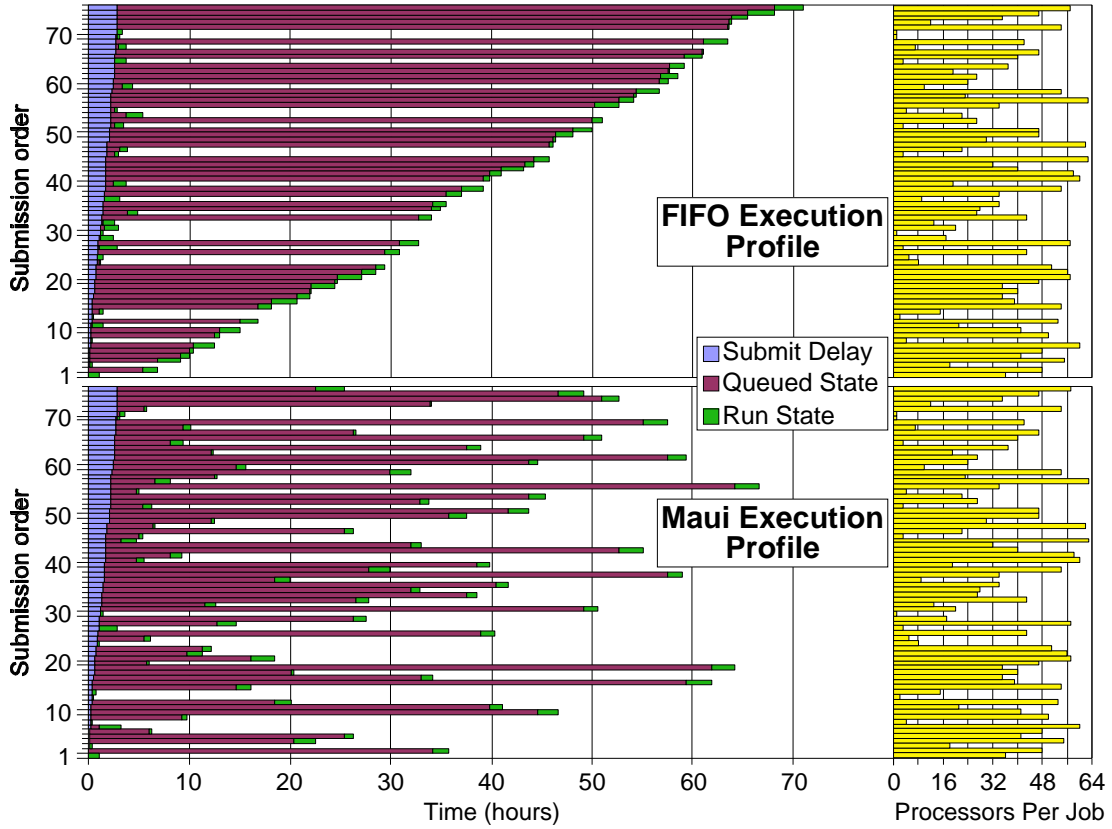
Figure 2. The execution profiles for the FIFO and Maui batch queues are presented in the left bar chart showing the submission delay, the wait time, and the run time respectively for each job. The right panel shows the number of processors requested by each of these jobs when they execute.

A further analysis of the data reveals that the FIFO scheduler starts all of the jobs with fewer than 10 nodes requested within 1 hour of submission. On the other hand Maui starts the last job with fewer than 10 nodes after over 16 hours in the queue. This difference is significant because it allows Maui to overlap the execution of more jobs during the test run, than does the FIFO scheduler. Indeed while the FIFO scheduler produces queue wait times nearly independent of the number of processors, ignoring the small jobs, the queue wait times under Maui are more similar to a bell curve with the maximum wait times experienced by jobs with node requests of approximately half the number of available nodes.

6.2 Theoretical Simulated Job Mix Results

In order to better evaluate the different schedulers performance for the evaluation job mix, a simulation routine was implemented that determined the first in first out (FIFO) execution of an ordered series of jobs. All jobs are executed in the specified order filling the system

to the maximum number of nodes (e.g., 64). By running this routine with the submit order of the 76 simulation jobs, the FIFO execution time would be 69.63 hrs. Executing them in reverse order gives a FIFO execution time of 70.46 hrs. The jobs ordered as they were run in both the PBS/FIFO and PBS/MAUI simulations yields 69.16 hrs and 65.99 hrs, respectively. This demonstrates that the delayed submission does
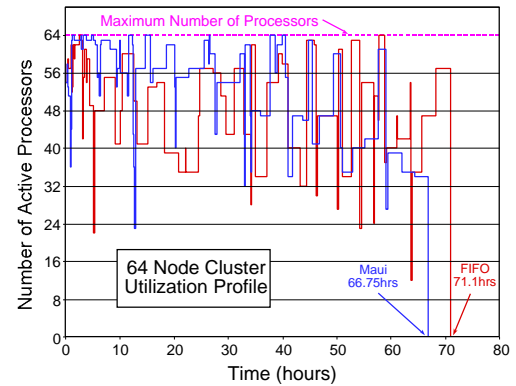

Figure 3. Cluster utilization comparison for PBS with FIFO, and with the Maui Scheduler active.

have an effect on how the jobs are eventually executed. In theory, with this routine we could find the optimal order for this specific set of jobs. Since there are 76 factorial (76!) possible orders, we did not pursue this.

6.3 Real Job Mix

The alpha cluster, running under moderately loaded conditions, has averaged 78% node hour utilization over the past three months. This was achieved while exceeding a total of over 1,200 jobs with node usage between 1 and 16 CPUs (Ave. of 4.3) and up to 2 wall days of time, the queue maximum. Out of the 1,200 jobs only 65 experienced a queue wait time of more than one day and most, 880, waited less than one hour to start execution.

This performance is despite the fact that the users are very poor at predicting the run time of their jobs. In fact the vast majority, 1146, of the jobs simply requested the maximum queue run time (2 days). The resulting job efficiencies were quite poor with only 140 jobs having an efficiency greater than 50% (i.e. using more than half of their requested time). It is unlikely that the job efficiencies will improve unless the load on the cluster increases producing longer queue wait times. Without long queue wait times users do not have much incentive to accurately predict the job run time or to attempt to fit a job into an existing hole in the job backfill window.

7. Future Directions

While we feel that the Maui Scheduler does an excellent job of scheduling jobs on flat interconnected clusters, a major area of on-going research is locality based scheduling. That is, scheduling based upon the topology of the interconnect, which might include interconnects with a tree structure and will certainly include SMP building blocks. This type of scheduling will become even more important in the near future since it becomes increasingly difficult and expensive to build a flat interconnect as the cluster size grows. In addition, new interconnect technologies are appearing which use loop, mesh and torus topologies.

There are of course many other areas of job resource management that need improvement on clusters. For example, job startup, monitoring and cleanup should be done in a parallel fashion. In addition the database of node and job status needs significant work to handle large clusters with large numbers of jobs effectively.

We plan to augment the simulations here with several techniques. First instead of a single pre-defined list of jobs randomly generated from a single source we will use "user-agents" that will submit jobs to the system. Each user-agent will submit jobs randomly generated but from a sub-class of the overall job mix. For example, a user-agent might represent a code developer submitting many debug jobs during normal working hours, a heavy user that submits long jobs, a greedy user that submits many jobs which fill gaps that a backfill mechanism might recognize, etc. The second modification is to change the metric. That will become the total number of active node hours in a given fixed time length. The user-agents will stop submitting jobs only after the metric has been met.

8. Conclusions

We have shown that the Maui Scheduler plugin to the PBS package provides a significant improvement in overall cluster utilization compared with the built-in FIFO scheduler. The Maui Scheduler does this by combining an intelligent wall time based node reservation system with an efficient backfill algorithm. The result is a flexible policy based scheduling system that provides guaranteed maximum start times while maintaining high total node utilization.

There are many issues that have yet to be addresses, such as cluster queue aggregation, inter-site trust and delayed authentication in addition to scalable system monitoring over a large distributed system.

9. References

[1] S. M. Jackson "QBank, A Resource Allocaiton Management Package for Parallel Computers, Version 2.6" (1998), Pacific Northwest National Laboratory, Richland, Washington 99352-0999.

[2] www.extremelinux.org/activities/usenix99/docs/

[3] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. J. Su, T. L. Windus, M. Dupuis,

J. A. Montgomery J.Comput.Chem. **14**, 1347-1363(1993)
[4] PBS_1, http://www.pbspro.com/
[5] http://www.mhpcc.edu/maui
[6] NERSC, The National Energy Research Scientific Computing Center, http:/www.nersc.gov
[7] This complete data set is provided to allow the reader to reproduce our simulations if desired: www.scl.ameslab.gov/Personnel/rickyk/jobmix.html